W W

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/520,405	03/08/2000	Michael G. Martinek	307.029US1/PA0390	1300

7590 09/08/2003

Mark A Litman & Associates PA
York Business Center Suite 205
3209 West 76th St
Edina, MN 55435

EXAMINER	
ASHBURN, STEVEN L	
ART UNIT	PAPER NUMBER
3714	22

DATE MAILED: 09/08/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

CS

Office Action Summary	Application No.	Applicant(s)
	09/520,405	MARTINEK ET AL.
	Examiner	Art Unit
	Steven Ashburn	3714

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *24 June 2003* .

2a)□ This action is **FINAL**. 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-38 and 47-57* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-38 and 47-57* is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)□ The specification is objected to by the Examiner.

10)□ The drawing(s) filed on _____ is/are: a)□ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)□ The proposed drawing correction filed on _____ is: a)□ approved b)□ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)□ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____ .

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

14)□ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) □ The translation of the foreign language provisional application has been received.

15)□ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) □ Notice of References Cited (PTO-892)	4) □ Interview Summary (PTO-413) Paper No(s). _____ .
2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) □ Notice of Informal Patent Application (PTO-152)
3) □ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ .	6) □ Other: .

DETAILED ACTION

Claim Rejections - 35 USC § 112

The rejection of claim 57 rejected under 35 U.S.C. 112, second paragraph, is withdrawn.

Claim Rejections - 35 USC § 103

Claims 1-3, 7, 11, 12, 16, 34-37, 48 and 52 are rejected under 35 U.S.C. 103(a) as being unpatentable over Bunnell, U.S. Patent 6,075,939 (Jun. 13, 2000).

Bunnell discloses a universal operating system with a modular kernel customizable to perform specialized applications. *See fig. 1, 2; 2:66-3:3.* The system's modularity allows it to be customized to support a range of applications ranging from small, embedded-type systems to large, network-capable instantiations. *See col. 2:66-4:2.* As a result, a developer may incorporate whatever optional kernel components desired, including incorporating application programs to maximize the system's resources by reducing computational overhead. *See id.*

In regards to claims 1 and 48: *Bunnell* describes the following features of the claimed invention:

a) A computerized controller having a processor, memory and nonvolatile storage. *See fig. 2.*

b) System handler, executed by the operating system kernel, operable to dynamically link with at least on a program object. *See fig. 1; 3:21-58.*

c) Application Program Interface (API) callable from the program object. *See col. 9:51-54.*

d) Operating system kernel that executes the system handler application. *See fig. 2; 3:21-58.*

Hence, *Bunnell* describes all the features of the claimed operating system except a "game" controller operable to control a "wagering game" by processing "gaming" program objects. Regardless of the deficiencies, applying the operating system described by *Bunnell* in a gaming device would have been obvious to an artisan.

Gaming systems are specialized applications that employ processors for executing wagering programs. Analogous controllers are widely used to control commercial devices having specialized functions including, for example, point-of-sale and security systems. *See, e.g., col. 8:16-19.* A gaming artisan would possess knowledge of these analogous processors for controlling specialized applications because they are pertinent to the problems of controlling specialized, commercial systems. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the operating system taught by *Bunnell* to apply in a "game" controller operable to control a "wagering game" by processing "gaming" program objects to provide an improved wagering game having with a modern operating system having the features of stability, configurablilty, minimal hardware requirements, low system management needs, high performance, real time support, low operating overhead, and be adaptability. *See col. 1:26-2:63.*

In regards to claim 2, *Bunnell* additionally teaches a system handler comprising a plurality of device handlers. *See figs 1-5.*

In regards to claim 3, *Bunnell* additionally teaches a system handler unloading, loading and executing program objects. *See fig. 1-5; See 3:43-59; 9:50-54; 11:20-12:15.*

In regards to claim 7, *Bunnell* describes all the features of the claims except using a pc-based system. Regardless, it was notoriously well known at the time of the invention to employ PC-based systems in custom applications because of their cost, availability, flexibility, compatibility and broad commercial support. Furthermore, it was well known in the gaming art at the time of the invention to employ personal computers as game controllers. Thus, it would have been obvious to a gaming artisan at the time of the invention to modify the operating system suggested by *Bunnell* to implement gaming systems with IBM PC-compatible controllers to decrease the cost and increase the supportability of a gaming device.

In regards to claim 11, *Bunnell* additionally teaches a system handler comprising a plurality of API's with a library of functions callable from program objects. *See fig. 4; col. 3:43-59, 9:50-54, 11:20-12:15.*

In regards to claim 12, *Bunnell* additionally teaches a system handler managing an event queue determining the order of the device handlers. *See fig. 1, 2; 3:39-42; 5:26-46, 6:26-47.*

In regards to claims 16, *Bunnell* additionally teaches loading a first program object which calls a function from within an API. *See fig. 2, 4.*

In regards to claims 34 and 36 and 37, *Bunnell* discloses a customizable operating system with an event queue, however it does not describes the specifics of the queue management. Thus, *Bunnell* describes all the features of the instant claims except the event queue queuing on first-come, first-server basis *(claim 36) and* the event queue queuing using more than one criteria *(claim 37).* Regardless of the deficiencies, the features were known in the art at the time of the invention and would have been obvious to an artisan.

Event queues are a basic function of an operating system for managing the system's response to events. It is fundamental programming technique to arrange the event response protocol to respond to events in the most effective order. For example, in some cases, event queues are simply organized on a first-serve/fist-serve basis. In other cases, the some events are more critical than others. Thus, the event queues are simply organized on a priority basis. In still other cases, events are handled on both a priority and first-come/first-serve basis. It would be a matter of design choice as to which manner the event queue managed events. Thus, it would have been obvious to an artisan at the time of the invention to modify the gaming operating system suggested by *Bunnell* to manage to event queuing on first-come, first-server basis or using more than one criteria to manage the event queue's priorities to respond to common events, such as button presses, on a first-come/first-serve basis while responding to critical events, such as security faults, immediately based on a higher priority.

In regards to claim 35: *Bunnell* additionally teaches a system handler comprising a plurality of API's with a library of functions callable from program objects. *See fig. 4; col. 3:43-59, 9:50-54, 11:20-12:15.*

In regards to claims 52, the operating system described by *Bunnell* teaches or suggests every feature of the claim except dynamically linking an API from a system from the system handler. Regardless, it is notoriously well known to dynamically link functions or data to a program during execution to reduce the resources used by a system by only linking when the object is required by a program or to provide for the addition or deletion of new components. Furthermore, if the did not dynamically link than the system would be static and require recompilation whenever a new component was added or removed and thereby provide. Thus, in this case, wherein the system is an embedded controller in a specialized application such as gaming machine, it would be obvious to an artisan at the time of the invention to modify the operating system described by *Bunnell*, wherein the operating system is customizable for specialized applications such as gaming, to dynamically link an API from a system from the system handler to provide a more robust, scalable, flexible operating system.

Claims 13, 15, 17-19, 21-27, 31, 34, 36, 37, 49, 53 and 54 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of Fullerton, U.S. Patent 4,607, 844 (Aug. 26, 1986).

In regards to claim 49, *Bunnell* describes all the features of the claim except storing data in non-volatile storage to preserve the state of a game in case of a loss of power. *Fullerton* discloses a gaming machine that stores data variables in non-volatile storage to increase security due to tampering or loss of power. *See col. 1:5-2:41.* In view of *Fullerton*, it would have been obvious to a gaming artisan at the time of the invention to modify the operating system describe by *Bunnell*, wherein the operating system is customizable for specialized applications, to modify the system to store game-state variables in non-volatile storage in case of loss of power to increase security and allow the games to be recovered.

In regards to claims 13, 19, 22, 24, 25, 27: *Bunnell* additionally teaches the following features:

a. Causing a system handler application to load and execute program objects. *See fig. 2-5.* More specifically, a primary purpose of a system handler serves in an operating system to load and execute program objects. Hence, although *Bunnell* does not explicitly describe the feature, it is implicit in *Bunnell's* operating system.

b. Causing a loaded program object to call-up a library of functions provided by the system handler. *See id.* More specifically, a fundamental purpose of a system handler in an operating system is to simplify calls from programs to storage devices to access resources such as function libraries. Hence, although *Bunnell* does not explicitly describe the feature, it is implicit in *Bunnell's* operating system.

c. Load a first program object from the library, execute the first program object, unload the first program object and load the second program object. *See id.* A fundamental purpose of an operating system is to manage the device's physical memory. In executing a program, the operating system is required to place program objects in memory to execute them, and subsequently remove the program object to swap-in a second program object for execution. Hence, the claimed feature is implicit in *Bunnell's* operating system.

d. Store data variables in storage such that a second program object in the library later loaded can access the data variables in storage. *See fig. 2-5.*

e. Loading a first program object which calls a function from within an API. *See fig. 2, 4.*

In regards to claims 15 and 21, *Bunnell* additionally teaches handling events via the system handler. *See col. 3:25-30, col. 5:32-46, 9:6-37.*

In regards to claim 17, *Bunnell* additionally teaches loading a first program object which calls a function from within an API. *See fig. 2, 4.*

In regards to claims 18 and 25, *Bunnell* additionally teaches load a first program object from the library, execute the first program object, unload the first program object and load the second program object. *See id.* A fundamental purpose of an operating system is to manage the device's physical memory. In executing a program, the operating system is required to place program objects in memory to execute them, and subsequently remove the program object to swap-in a second program object for execution. Hence, the claimed feature is implicit in *Bunnell's* operating system.

In regards to claims 23 and 26, it is implicit in the operating system describe by *Bunnell* that instructions are operable when executed to cause a computer to provide functions through an API that comprises part of the system handler application because a fundamental function of a system handler is to provide an API to simplify functions calls by applications to the kernel.

In regards to claims 31, *Bunnell* additionally teaches a system handler comprising a plurality of device handlers. *See figs 1-5.*

In regards to claims 7 and 29, *Bunnell* describes all the features of the claims except using a pc-based system. Regardless, it was notoriously well known at the time of the invention to employ PC-based systems in custom applications because of their cost, availability, flexibility, compatibility and broad commercial support. Furthermore, it was well known in the gaming art at the time of the invention to employ personal computers as game controllers. Thus, it would have been obvious to a gaming artisan at the time of the invention to modify the operating system suggested by *Bunnell* to implement gaming systems with IBM PC-compatible controllers to decrease the cost and increase the supportability of a gaming device.

In regards to claim 34: *Bunnell* additionally teaches a system handler managing an event queue determining the order of the device handlers. *See fig. 1, 2; 3:39-42; 5:26-46, 6:26-47.*

In regards to claims 36 and 37, *Bunnell* discloses a customizable operating system with an event queue, however it does not describes the specifics of the queue management. Thus, *Bunnell* describes all the features of the instant claims except the event queue queuing on first-come, first-server basis *(claim 36) and* the event queue queuing using more than one criteria *(claim 37).* Regardless of the deficiencies, the features were known in the art at the time of the invention and would have been obvious to an artisan.

Event queues are a basic function of an operating system for managing the system's response to events. It is fundamental programming technique to arrange the event response protocol to respond to events in the most effective order. For example, in some cases, event queues are simply organized on a first-serve/fist-serve basis. In other cases, the some events are more critical than others. Thus, the event queues are simply organized on a priority basis. In still other cases, events are handled on both a priority and first-come/first-serve basis. It would be a matter of design choice as to which manner the event queue managed events.

Thus, it would have been obvious to an artisan at the time of the invention to modify the gaming operating system suggested by *Bunnell* to manage to event queuing on first-come, first-server basis or using more than one criteria to manage the event queue's priorities to respond to common events, such as button presses, on a first-come/first-serve basis while responding to critical events, such as security faults, immediately based on a higher priority.

In regards to claims 53 and 54, the operating system described by *Bunnell* teaches or suggests every feature of the claim except having instructions, when executed, operable to dynamically link an API to a program object. Regardless, it is notoriously well known to dynamically link functions or data to a program during execution to reduce the resources used by a system by only linking when the object is required by a program. Thus, it would have been obvious to an artisan at the time of the invention to modify the operating system described by *Bunnell*, wherein the operating system is customizable for specialized applications such as gaming, to dynamically link an API to a program object to reduce the resources used by a system and thereby increase efficiency.

Claims 4, 5 and 55-57 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Fullerton*, as applied to claims 49 above, in further view of Arbaugh et al., U.S. Patent 6,185,678 B1 (Feb 6, 2001).

In regards to claim 4 and 55-57 the combination of *Bunnell* and *Fullerton* describes all the features of the claims except having the operating system verify that the operating system kernel or code for a shared object has not changed. Regardless of the deficiency, this feature would have been obvious to an artisan in view of *Arbaugh*.

Arbaugh discloses an method for initializing a computer system that ensures a system's integrity. *See col. 4:33-59.* The reference teaches that it is known in prior system to verify software upon boot-up and while the operating system is running. *See col. 2:52-3:3.* Integrity is validated at each layer transition in the bootstrap process and a recovery process is included for integrity check failures. Ensuring the integrity is provided by the use of public key cryptography, a cryptographic hash function, and public key certificates. *See id.* The system does this by constructing a chain of integrity checks, beginning at power-on and continuing until the final transfer of control from the bootstrap components to the operating system itself. *See id.* The integrity checks compare a computed cryptographic hash value with a stored digital signature associated with each component. *See id.* Cryptographic algorithms are combined with the chosen protocols to add security to the recovery process, however if security is not a concern, then a less robust approach could be used. *See id.* By ensuring the integrity of the system, *Augaugh* suggests performing integrity checks improves a system's security, reliability and total ownership cost will be improved. *See col. 4:33-37, 4:60-65.*

In view of *Arbaugh*, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the operating system suggested by *Bunnell* and *Fullerton*, wherein a specialized operating system protects data by storing it in non-volatile memory, to add the feature of having the operating system verify that the operating system kernel or code for a shared object has not changed to improve the data processing systems security, integrity, reliability and total ownership cost.

In regards to claim 5, the combination above describes all the claimed features except crating a list of pointers that associate variable names with data locations. Regardless of is notoriously well known in the art of data processing to create a list of pointers that associate variable names with data locations such that a process can locate data by referencing variables. Thus, it would have been obvious to an artisan at the time of the invention to add the feature of creating a list of pointers that associate variable names with data locations such that a process can locate data by referencing variables and thereby make programming simpler by not having to specifically reference memory addresses.

ClaimS 6, 14 and 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Fullerton*, as applied to claim 4 above, in further view of *Pascal*.

The combination of *Bunnell* with *Fullerton* describes all the features of the claim except causing the execution of a corresponding callback function when a data variable is changed in non-volatile storage. Regardless of the deficiency, this feature would have been obvious to an artisan in view of *Pascal*.

Pascal discloses an analogous operating system for a gaming device wherein callbacks are employed to communicate information between application modules upon the occurrence of certain events. *See 1:44-2:30.* In generally, callback routines are used in state-based machines to communicate data between independent modules upon the occurrence of predetermined events. *See col. 6:25-45.* *Pascal* describes using callback to enhance to robustness of a gaming device under fault conditions to protect data that may affect the outcome of a game payout. *See col. 2:25-30.*

In view of *Pascal*, it would have been obvious to an artisan at the time of the invention to modify the customized gaming operating system suggested by the combination of *Bunnell* with *Fullerton*, wherein the system enhance security by monitoring application modules, to execute a callback function corresponding to a change in game data stored in non-volatile memory to enhance the security of the gaming device by monitoring changes in data that might affect the outcome of the game payout and thereby provide a more secure gaming device that is resistant to errors caused by losses in power or tampering.

Claims 8-10 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of David A. Rusling, *The Linux Kernel*, <http://www.tldp.org/LDP/-tlk/tlk.htm> (1999) (hereinafter "*Rusling*").

In regards to claim 8, *Bunnell* teaches all the features of the claim except using a LINUX operating system kernel. Regardless of the deficiency, this feature would have been obvious to an artisan.

Rusling describes the features of the LINUX operating system. The system is a well-known, freely distributed, PC-compatible operating system derived from UNIX. Analogously to *Bunnell*, the LINUX kernel is scalable and customizable for specialized applications. A gaming artisan at the time of the invention would possess knowledge of the fundamental features and relative advantages of the various operating systems, including LINUX. Thus, it would have been obvious to gaming artisan at the time of the invention to modify *Bunnell's* operating system to be based upon a LINUX foundation and thereby improve the gaming device by using an operating system customized for gaming using a scalable, free and widely supported operating system.

In regards to claims 9 and 10, *Bunnell* additionally teaches a modular kernel modifiable to support specialized applications by disabling unnecessary components including device handlers. *See col. 2:66-4:2, 13:34-43.* Hence it suggests disabling keyboard, I/O, network, and storage device handler when not required in an application.

Claims 38 and 55-57 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of Arbaugh et al., U.S. Patent 6,185,678 B1 (Feb 6, 2001).

Bunnell describes all the features of the claims except having the operating system verify that the operating system kernel or code for a shared object has not changed. *Arbaugh* discloses an method for initializing a computer system that ensures a system's integrity. *See col. 4:33-59.* The reference teaches that it is known in prior system to verify software upon boot-up and while the operating system is running. *See col. 2:52-3:3.* Integrity is validated at each layer transition in the bootstrap process and a recovery process is included for integrity check failures. Ensuring the integrity is provided by the use of public key cryptography, a cryptographic hash function, and public key certificates. *See id.* The system does this by constructing a chain of integrity checks, beginning at power-on and continuing until the final transfer of control from the bootstrap components to the operating system itself. *See id.* The integrity checks compare a computed cryptographic hash value with a stored digital signature associated with each component. *See id.* Cryptographic algorithms are combined with the chosen protocols to add security to the recovery process, however if security is not a concern, then a less robust approach could be used. *See id.* By ensuring the integrity of the system, *Augaugh* suggests performing integrity checks improves a system's security, reliability and total ownership cost will be improved. *See col. 4:33-37, 4:60-65.*

In view of *Arbaugh*, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the operating system suggested by *Bunnell*, wherein a specialized operating system protects data by storing it in non-volatile memory, to add the feature of having the operating system verify that the operating system kernel or code for a shared object has not changed to improve the data processing systems security, integrity, reliability and total ownership cost.

Claim 47 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Pascal*, as applied to the claims 22, 23 above, in further view of *Bock*, U.S. Patent 5,155,856 (Oct. 13, 1992) and *Davis*, U.S. Patent 6,401,208 (Jun. 4, 2002).

Bunnell discloses a scalable, modular operating system that selectively initializes only the parts of the kernel required under a system profile and does not attempt to initialize components that are not present. *See col. 11:20-28, 12:17-23, 13:3-12.* As a result, the system advantageously allows the operating system to configure kernel components in or out of the system. *See id.* Hence, *Bunnell* describes a system that disables selected device handlers.

Bunnell additionally discloses that the operating system can incorporate user-level code on several levels. For example, in a fully stand-alone embodiment, where only a single user-level application is executed, the application may be coupled directly with the operating system to allocate all the system resources to executing the single application. *See col. 8:39-46.* Accordingly, *Bunnell* does not limit what applications the developer includes within the kernel. *See 8:47-56.* Hence, *Bunnell* suggest coupling a user-level application with the operating system that is executed upon bootstrapping the system.

Pascal describes the process of executing an operating system from ROM upon bootstrapping a gaming device from a cold or warm start. *See fig. 2; col. 3:41-67.*

Thus, the operating system for a gaming device suggested by the combination of *Bunnell* with *Pascal* describes all the features of the present claims except zeroing-out unused RAM and testing and/or hashing the kernel. Regardless of the deficiencies, these features were known in the art at the time of the invention and would have been obvious to an artisan in view of *Bock* and *Davis*.

Bock describes "self-guarding" computer system that, upon initialization, zeros-out unused memory to erase information which is not required for subsequent operation. *See col. 1:38-62.* Erasing the data serves to eliminate vestigial data remaining in memory after reset that may interfere with the newly executed instructions and cause errors. *See col. 3:10-20.* Hence *Bock* suggests enhancing the security of a computing system by zeroing-out unused memory.

Davis describes a computing system that, upon initialization, tests and/or hashes the BIOS to enhance to system's security. The reference describes several threats to computing device that circumvent security measures implemented at higher levels of operation. For example, a malefactor may replace the device's ROM or a software virus may infiltrate the BIOS. In solution of this threat, *Davis* suggests testing the BIOS using a hash-function upon initialization. *See col. 1:32-2:5.*

The above references analogously describe means for initializing and increasing the security of an operating system. In view of *Bock* and *Davis*, it would have been obvious to a gaming artisan at the time of the invention to modify the gaming device operating system suggested by the combination of *Bunnell* with *Pascal* to add the features of zeroing-out unused RAM and testing and/or hashing the kernel. The modification would enhance the security of the gaming device by frustrating attempts to tamper with the operating system by modifying the data stored within the device's memory prior to initialization.

Claims 50 and 51 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Wiltshire*, U.S. Patent 6,409,602 (Jun. 25, 2002).

Bunnell teaches the operating system is scalable to be network compliant. *See col. 2:27-3:3.* As describes above, the reference describes all the features of the claimed subject matter except using the operating system to control a networked online system *(claim 45)* and using the system to control a progressive meter *(claim 46)*. Regardless of the deficiencies, the features were known in the art at the time of the invention and would have been obvious to an artisan in view of *Wiltshire*.

Wiltshire discloses an analogous gaming system in which PC-based machines execute commercially available operating systems to decrease the cost of developing and upgrading gaming devices. *See col. 2:6-44, 4:44-7:5.* The reference describes an operating system controlling a networked, online gaming system including managing a progressive jackpot. *See col. 4:66-5:13, 5:45-64.*

In view of *Wiltshire*, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the gaming operating system suggested by *Bunnell,* wherein the operating system is scalable to support networks, to add the features of controlling a networked online system and controlling a progressive meter. The modification would allow the gaming device to communicate over a casino networks and thereby make the device attractive to operators whose casinos link gaming machines to networks for player tracking, accounting, monitoring, and linking to progressive jackpots.

Claims 28-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Houriet, Jr. et al., U.S. Patent 5,575,717 (Nov. 19, 1996) in view of Mitchell et al., U.S. Patent 5,872,973 (Feb. 16, 1999).

In regards to claim 28, *Houriet* discloses a system for allowing a game operator to individually tailor parameters of a video game machine provides a plurality of user interactive video screen displays for selecting the parameters. The parameters include the selection of menu choices of video games from a library of video games that are available for play on the machine. *See abstract.* In particular regards to the claim, *Houriet* teaches a gaming device incorporating a program containing a large number of games wherein a mode selector allows selection of the game to be executed by the device. *See col. 1:61-2:36.* Furthermore, it is inherent that electronic controller that operates the electronic gaming system includes an operating system for controlling the hardware and software that comprises the game. Thus, *Houriet* teaches a gaming machine comprising an operating system and a plurality of game programs describing a game personality is a selected mode. However, the reference does not describe a plurality of shared objects wherein each shared object describes the game personalities. Regardless of the deficiency, this feature would have been obvious to an artisan.

Mitchell a system for creating named relations between classes in a dynamic object-oriented programming environment. The objects dynamically bind to the class interfaces of the classes being related. The relationships can be programmatically attached by name to object instances during program execution. Because these relationships are stored in a resource and are dynamically bound by name to the objects, they can be created and modified without requiring the source code of the objects being associated to be changed. This eliminates hard coded dependencies between objects that impede reuse of the objects in other contexts. In particular regards to the claim, *Mitchell* teaches that it was known in the art to structure programs into sharable modules that are linked to form complex programs. *See col. 1:24-2:57, 4:30-5:19.* As a result, program development becomes simpler and less costly because modules can be reused instead of recreated each time. *See id.* Thus, it was known at the time of the invention to form a program from a plurality of shared objects wherein each describes features of the program.

In view of *Mitchell*, it would have been obvious to an artisan at the time of the invention to modify the gaming system described by *Houriet*, wherein the device contains a plurality of selectable game modes having describing a game personality is a selected mode, to add the feature of shared objects wherein each shared object describes the game personalities. As taught by *Mitchell*, the modification would enhance the system by making program development simpler and less costly because modules can be reused instead of recreated. *See id.*

In regards to claim 29, *Mitchell* additionally describes an operating system based on an IBM-PC. *See col. 4:30-49.*

In regards to claim 30, *Houriet* describes a gaming device wherein an electronic controller responds to events such as user inputs to a touchscreen display. *See fig. 4; col. 3:33-57.* Hence, the reference includes an operating system with software capable of detecting and responding to events. However, *Houriet* does not explicitly describe a system handler. Regardless of the deficiency, this feature would have been obvious to an artisan.

A system handler is a virtual device created in software that provides an interface between application programs and machine-level devices in order to automate the task of exchanging data and thereby reduce to complexity and effort required to write applications by allowing higher-level applications to call upon lower-lever services. System handlers typically respond to events detected by the system. For example, in a gaming device, an event requiring a response would allow a security application to receive a signal from hardware indicating the opening of access panel on the housing of the gaming device causing the event to be logged in storage and activating an indicator. Thus, it is known in the art to employ system handlers operating systems for gaming devices.

In regards to claim 31, as described above, *Houriet* describes all the features of the claims except a device handler. Regardless of the deficiency, this feature would have been obvious to an artisan.

A device handler is a virtual device created in software that provides an interface between application programs and hardware devices in order to automate the task of exchanging data and thereby reduce to complexity and effort required to write applications. For example, in a gaming device, an event requiring a response would include opening an access panel to allow the event to be logged in storage and activate an indicator. Thus, it is known in thc art to employ device handlers in operating systems for gaming devices.

Consequently, it would have been obvious to an artisan at the time of the invention to modify the gaming system described by *Houriet*, wherein a operating system controls a electronic processor, to add the feature of a system handler to automate the task of exchanging data and thereby reduce to complexity and effort required to write applications.

In regards to claim 32, as described above, *Houriet* describes all the features of the claims except an event queue. Regardless of the deficiency, this feature would have been obvious to an artisan.

Scheduling is a task performed by a system handler to allow a computer to respond to events based on priority. Typically, a event queue is generated that schedules initiation of routines associated with the events. For example, the handling events such as touch-screen inputs would commonly be a higher priority than a door-open because of the faster response time required to by the event. Thus, it is known in the art to employ event queue in operating systems for gaming devices. Consequently, it would have been obvious to an artisan at the time of the invention to modify the gaming system described by *Houriet*, wherein a operating system controls a electronic processor, to add the feature of a event queue to allow the system to respond to events based on priority.

In regards to claims 33, *Mitchell* additionally describes a plurality of API callable functions. *See col. 1:24-2:57.*

Response to Arguments

Applicant's arguments filed June 24, 2003 have been fully considered but they are not persuasive.

Response to applicant's preliminary remarks

The applicant argues that the independent claims distinguish over the prior art because *Bunnell* does not describe the claimed feature of an active program objects that call a system handler application. Furthermore, the applicant asserts that this feature distinguish from *Bunnell* because the program objects described in the reference are passive, not active. Moreover, the applicant asserts that the claims distinguish from *Bunnell* because the reference does not teach the "order of operation and execution" of functions required by the claims. The examiner respectfully disagrees for several reasons.

First, the features asserted in the applicant's arguments are not described in the claims. Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993). In this case claim 1, for example, requires (i) an operating system comprising a system handler application operable to dynamically link with at least one program object; (ii) an API having functions callable from a program object; and (iii) an operating system kernel that executes the system handler application. The claim does not include the limitation of "active" program objects that call a system handler application. Furthermore, the claim does include any language defining an "order of operation and execution" of functions (e.g. first, second, next, then).

Second, the applicant's argument appears contradictory. The applicant first asserts that claimed program objects make calls to a system handler. This suggests that the program objects are bypassing the API. However, the applicant asserts next that the claimed program objects are distinguished because they access the API.

Third, *Bunnell* teaches or suggests the features argued by the applicant. The reference describes program objects actively accessing system handler applications through an API. *See fig. 4; col. 3:24-29; 9:7-16, 9:45-54, 11:52-56.* Alternatively, the system can bypass the API by making program objects part of the kernel. *See col. 8:57-64.*

Consequently, for all the reasons given above, the applicant's arguments are unpersuasive.

Response to applicant's arguments of claims 1-38 and 48-57

The applicant asserts that the examiner's rejection fails to appreciate the complexity of a system in a gaming system environment. In response the examiner respectfully notes that the breadth and brevity of the applicant's claims do not describe a complex system.

Furthermore, the applicant asserts that the examiner's rejection fails to appreciate the difference between general PC usage and usage in a gaming system. In response, *Bunnell*, the reference mainly relied on in the rejection does not describe a general PC. Instead, *Bunnell* discloses a universal operating system with a modular kernel customizable to perform specialized applications. *See fig. 1, 2; 2:66-3:3.* The system's modularity allows it to be customized to support a range of applications ranging from small, embedded-type systems to large, network-capable instantiations. *See col. 2:66-4:2.* As a result, a developer may incorporate whatever optional kernel components desired, including incorporating application programs to maximize the system's resources by reducing computational overhead. *See id.*

Still furthermore, the applicant argues that the *Bunnell* is not analogous to the claimed invention. The examiner respectfully disagrees. A prior art reference must either be in the field of applicant's endeavor or, if not, then be reasonably pertinent to the particular problem with which the applicant was concerned. See *In re Oetiker*, 977 F.2d 1443, 24 USPQ2d 1443 (Fed. Cir. 1992). In this case, *Bunnel* is reasonably pertinent to the particular problem which the applicant is concerned. The applicant identifies several problems to which the invention is directed. These include (i) lack of standardization between gaming devices due, in part, to reliance on primitive operating systems *(see p. 4, lines 16-21)*; (ii) lack of uniformity in hardware interfaces *(see p. 5, lines 3-14)*. To solve these problems, the applicant provides a "universal controller" tailored for use in gaming devices to improve the efficiency, security and operational costs. In comparison, *Bunnell* discloses a "universal" operating system for use in embedded systems that can be scaled and reconfigured to support specialized computing applications and thereby reduce the time required to retrofit or modernize an existing operating system. *See col. 1:7-38, 2:59-64.* Thus, *Bunnell* is reasonably pertinent to the problem of providing an ""universal controller" wherein the operating system for the specialized application of gaming.

Still furthermore, the applicant argues that the claims distinguish from *Bunnell* because the reference does not teach dynamic linking in the manner required by the claims. More specifically, the applicant asserts that the claimed system performs dynamic linking at a different abstract level of software (e.g. the application level) than described in *Bunnell*. The examiner respectfully disagrees. First, the examiner does not find a limitation argued expressed in the claims. Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993). Here, the claims neither sets forth different application levels nor expresses which level dynamic link occurs at. Second, *Bunnell* teaches the claimed feature of dynamic linking. *Bunnell* teaches dynamically linking system objects that, in an embedded system (such as a gaming system), may include program objects. *See 8:57-63.* Additionally, *Bunnell* teaches an operating system that dynamically links a system handler and program objects into the kernel upon boot-up. *See col. 3:25-58, 4:47-5:25, 6:25-54, 8:57-63.* Hence, *Bunnell* teaches the claimed feature. The examiner notes that *Rusling* describes the LINUX operating system in which program objects to be dynamically loaded and unloaded as they are needed after the system has booted. *See chapter 12.*

Still furthermore, the applicant argues that the claims are fundamentally different than *Bunnell* and the other prior art of record because the references do not describe the limitation that program objects are required to call functions from within the API. The examiner respectfully disagrees. First, the examiner does not find this limitation in the claims. Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993). Here, the claims require that the API has functions callable from a program object. *See, e.g., claim 1. Bunnell* teaches this feature. *See fig. 2, 4.* More specifically, the Figures 2 and 4 illustrate program object calling functions in the API level which subsequently call operating system functions. Hence, the argument unpersuasive.

Still furthermore, in regards to claims 48, 55-57 the applicant argues that *Bunnell* does not teach the claimed feature of verifying that an operations system kernel shared object has not changed as required. The examiner respectfully disagrees. First, claim 48 does not include this feature. Second, claims 55-57 are rejected by *Bunnell* in view of *Arbaugh.* In response to applicant's arguments against *Bunnell* individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.,* 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). Hence, the argument is unpersuasive.

Still furthermore, in regards to claims 5, 19, 21-23, 25 and 26, the applicant argues that *Bunnell* does not describe causing a system handler to load and execute program objects; cause a loaded object to call a library of functions, load an object from a library or store data variables in nonvolatile storage such that a second program object in the library can access the data. The examiner respectfully disagrees. First, claim 5 does not claim these features. Second, claims 21-23, 25 and 26 are rejected by the examiner by *Bunnell* in view of *Fullerton*. In response to applicant's arguments against *Bunnell* individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

The standard of patentability is what the prior art, taken as a whole, suggests to an artisan at the time of the invention. *In re Merck & Co., Inc., 800 F.2d 1091,1097, 231 USPQ 375, 379 (Fed. Cir. 1986)*. The question is not only what the references expressly teach, but what they would collectively suggest to one of ordinary skill in the art. *In re Simon, 461 F.2d 1387, 1390, 174 USPQ 114, 116 (CCPA 1972)*. In this case, *Bunnell* additionally teaches the following features:

f. Causing a system handler application to load and execute program objects. *See fig. 2-5.* More specifically, a primary purpose of a system handler serves in an operating system to load and execute program objects. Hence, although *Bunnell* does not explicitly describe the feature, it is implicit in *Bunnell's* operating system.

g. Causing a loaded program object to call-up a library of functions provided by the system handler. *See id.* More specifically, a fundamental purpose of a system handler in an operating system is to simplify calls from programs to storage devices to access resources such as function libraries. Hence, although *Bunnel* does not explicitly describe the feature, it is implicit in *Bunnell's* operating system.

h. Load a first program object from the library, execute the first program object, unload the first program object and load the second program object. *See id.* A fundamental purpose of an operating system is to manage the device's physical memory. In executing a program, the operating system is required to place program objects in memory to execute them, and subsequently remove the program object to swap-in a second program object for execution. Hence, the claimed feature is implicit in *Bunnell's* operating system.

i. Store data variables in storage such that a second program object in the library later loaded can access the data variables in storage. *See fig. 2-5.*

Fullerton discloses a gaming machine that stores data variables in non-volatile storage to increase security due to tampering or loss of power. See col. 1:5-2:41. Thus, when the prior art is taken as a whole by an artisan at the time of the invention, it collectively suggests an gaming device having software causing a system handler to load and execute program objects; cause a loaded object to call a library of functions, load an object from a library or store data variables in nonvolatile storage such that a second program object in the library can access the data. Consequently, the rejection is maintained.

Response to applicant's arguments of claims 9, 10, 17, 38 and 47

In further regards to claims 9, 10, 17, 38 and 47, the applicant argues that claimed invention distinguishes over the prior art because the combination of *Bunnell* in view of *Pascal, Bock* and *Davis* does not teach or suggest zeroing-out unused registers. The examiner respectfully disagrees.

In response to applicant's arguments against the references individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.,* 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). The standard of patentability is what the prior art, taken as a whole, suggests to an artisan at the time of the invention. *In re Merck & Co., Inc., 800 F.2d 1091,1097, 231 USPQ 375, 379 (Fed. Cir. 1986).* The question is not only what the references expressly teach, but what they would collectively suggest to one of ordinary skill in the art. *In re Simon, 461 F.2d 1387, 1390, 174 USPQ 114, 116 (CCPA 1972).*

In this case, *Bunnell* discloses a scalable, modular operating system that selectively initializes only the parts of the kernel required under a system profile and does not attempt to initialize components that are not present. *See col. 11:20-28, 12:17-23, 13:3-12.* As a result, the system advantageously allows the operating system to configure kernel components in or out of the system. *See id.* Hence, *Bunnell* describes a system that disables selected device handlers. *Pascal* describes the process of executing an operating system from ROM upon bootstrapping a gaming device from a cold or warm start. *See fig. 2; col. 3:41-67. Bock* describes "self-guarding" computer system that, upon initialization, zeros-out unused memory to erase information which is not required for subsequent operation. *See col. 1:38-62.* Erasing the data serves to eliminate vestigial data remaining in memory after reset that may interfere with the newly executed instructions and cause errors. *See col. 3:10-20.* Hence *Bock* suggests enhancing the security of a computing system by zeroing-out unused memory. *Davis* describes a computing system that, upon initialization, tests and/or hashes the BIOS to enhance to system's security. The reference describes several threats to computing device that circumvent security measures implemented at higher levels of operation. For example, a malefactor may replace the device's ROM or a software virus may infiltrate the BIOS. In solution of this threat, *Davis* suggests testing the BIOS using a hash-function upon initialization. *See col. 1:32-2:5.* The above references analogously describe means for initializing and increasing the security of an operating system. Hence, when the prior art is taken as a whole by a gaming artisan, it collectively suggests zeroing-out unused memory.

Consequently, for all the reasons given above, the rejection is maintained.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Steven Ashburn whose telephone number is 703 305 3543. The examiner can normally be reached on Monday thru Friday, 8:00 AM to 4:30 PM. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tom Hughes can be reached on 703-308-1806. The fax phone numbers for the organization where this application or proceeding is assigned are 703 872 9302 for regular communications and 703 872 9303 for After Final communications. Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703 308 1078.

s.a.
September 7, 2003

S. THOMAS HUGHES
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 3700